Juggling Functions Inside a Database

Mahmoud Abo Khamis
LogicBlox Inc.
mahmoud.abokhamis@logicblox.com

Hung Q. Ngo
LogicBlox Inc.
hung.ngo@logicblox.com

Atri Rudra
University at Buffalo, SUNY
atri@buffalo.edu

ABSTRACT
We define and study the Functional Aggregate Query (FAQ) problem, which captures common computational tasks across a very wide range of domains including relational databases, logic, matrix and tensor computation, probabilistic graphical models, constraint satisfaction, and signal processing. Simply put, an FAQ is a declarative way of defining a new function from a database of input functions.

We present InsideOut, a dynamic programming algorithm, to evaluate an FAQ. The algorithm rewrites the input query into a set of easier-to-compute FAQ sub-queries. Each sub-query is then evaluated using a worst-case optimal relational join algorithm. The topic of designing algorithms to optimally evaluate the classic multiway join problem has seen exciting developments in the past few years. Our framework tightly connects these new ideas in database theory with a vast number of application areas in a coherent manner, showing potentially that—with the right abstraction, blurring the distinction between data and computation—a good database engine can be a general purpose constraint solver, relational data store, graphical model inference engine, and matrix/tensor computation processor all at once.

The InsideOut algorithm is very simple, as shall be described in this paper. Yet, in spite of solving an extremely general problem, its runtime either is as good as or improves upon the best known algorithms for the applications that FAQ specializes to. These corollaries include computational tasks in graphical model inference, matrix/tensor operations, relational joins, and logic. Better yet, InsideOut can be used within any database engine, because it is basically a principled way of rewriting queries. Indeed, it is already part of the LogicBlox database engine, helping efficiently answer transactional database queries, graphical model inference queries, and train a large class of machine learning models inside the database itself.

1. INTRODUCTION
The following fundamental problems from diverse domains share a common algebraic structure involving (generalized) sums of products.

Example 1. (Matrix Chain Multiplication (MCM)) Given a series of matrices \( A_1, \ldots, A_n \) over some field \( F \), where the dimension of \( A_i \) is \( p_i \times p_{i+1} \), \( i \in [n] \), we wish to compute the product \( A = A_1 \cdots A_n \). The problem can be reformulated as follows. There are \( n + 1 \) variables \( X_1, \ldots, X_{n+1} \) with domains \( \text{Dom}(X_i) = [p_i] \), for \( i \in [n+1] \). For \( i \in [n] \), matrix \( A_i \) can be viewed as a function of two variables
\[
\psi_{i+1} : \text{Dom}(X_i) \times \text{Dom}(X_{i+1}) \to F,
\]
where \( \psi_{i+1}(x, y) = (A_i)_{xy} \). The MCM problem is to compute the output function
\[
\varphi(x_1, x_{n+1}) = \sum_{x_2 \in \text{Dom}(X_2)} \cdots \sum_{x_n \in \text{Dom}(X_n)} \prod_{i=1}^{n} \psi_{i+1}(x_i, x_{i+1}).
\]

Example 2. (Maximum A Posteriori (MAP) queries in probabilistic graphical models (PGMs)) Consider a discrete graphical model represented by a hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{E}) \). There are \( m \) discrete random variables \( \mathcal{V} = \{X_1, \ldots, X_m\} \) on finite domains \( \text{Dom}(X_i), i \in [n] \), and \( m = |\mathcal{E}| \) factors
\[
\psi_S : \prod_{i \in S} \text{Dom}(X_i) \to \mathbb{R}_+, \ S \in \mathcal{E}.
\]
A typical inference task is to compute the marginal MAP estimates, written in the form
\[
\varphi(x_1, \ldots, x_f) = \max_{x_{f+1} \in \text{Dom}(X_{f+1})} \cdots \max_{x_n \in \text{Dom}(X_n)} \prod_{S \in \mathcal{E}} \psi_S(x_S).
\]

Example 3. (Conjunctive query in RDBMS) Consider a schema with the following input relations: \( R(a, b), S(b, c), T(c, a) \), where for simplicity let us say all attributes are integers. Consider the following query:

\[
\text{SELECT R.a FROM R, S, T WHERE R.b = S.b AND S.c = T.c AND T.a = R.a;}
\]
The above query can be reformulated as follows. Relation \( R(a, b) \) is modeled by a function \( \psi_H(a, b) \to \{\text{true, false}\} \).
where $\psi_R(a, b) = \text{true}$ iff $(a, b) \in R$, and relations $S(b, c)$ and $T(c, a)$ are modeled by similar functions $\psi_S(b, c), \psi_T(c, a)$. Now, computing the above query basically corresponds to computing the function $\varphi(a) \rightarrow \{\text{true}, \text{false}\}$, defined as:

$$\varphi(a) = \bigvee_b \bigwedge_c \psi_R(a, b) \land \psi_S(b, c) \land \psi_T(c, a).$$

**Example 4. (# Quantified Conjunctive Query (#QCQ))** Let $\Phi$ be a first-order formula of the form

$$\Phi(X_1, \ldots, X_f) = Q_{f+1}X_{f+1} \cdots Q_nX_n \left( \bigwedge_{R \in \text{atoms}(\Phi)} R \right),$$

where $Q_i \in \{\exists, \forall\}$, for $i > f$. The #QCQ problem is to count the number of tuples in relation $\Phi$ on the free variables $X_1, \ldots, X_f$. To reformulate #QCQ, construct a hypergraph $\mathcal{H} = (V, E)$ as follows: $V$ is the set of all variables $X_1, \ldots, X_n$, and each $R \in \text{atoms}(\Phi)$ there is a hyperedge $S = \text{vars}(R)$ consisting of all variables in $R$. The atom $R$ can be viewed as a function indicating whether an assignment $x_S$ to its variables is satisfied by the atom; namely $\psi_R(x_S) = 1$ if $x_S \in R$ and 0 otherwise.

For each $i \in \{f+1, \ldots, n\}$ we define an aggregate operator $\Theta^{(i)} = \begin{cases} \max & \text{if } Q_i = \exists, \\ \times & \text{if } Q_i = \forall. \end{cases}$

Then, the #QCQ problem above is to compute the constant function

$$\varphi = \sum_{x_1 \in \text{Dom}(X_1)} \cdots \sum_{x_f \in \text{Dom}(X_f)} \Theta^{(f+1)}_{x_{f+1} \in \mathbb{0}, 1} \cdots \Theta^{(n)}_{x_n \in \mathbb{0}, 1} \psi_S(x_S).$$

It turns out that these and dozens of other fundamental problems from constraint satisfaction (CSP), databases, matrix operations, PGM inference, logic, coding theory, and complexity theory can be viewed as special instances of a generic problem we call the Functional Aggregate Query, or the FAQ problem, which we define next. (See [2, 6] for many more examples.)

Throughout the paper, we use the following convention. Uppercase $X_i$ denotes a variable, and lowercase $x_i$ denotes a value in the domain $\text{Dom}(X_i)$ of the variable. Furthermore, for any subset $S \subseteq [n]$, define

$$X_S = (X_i)_{i \in S}, \quad x_S = (x_i)_{i \in S} \in \prod_{i \in S} \text{Dom}(X_i).$$

In particular, $X_S$ is a tuple of variables and $x_S$ is a tuple of specific values with support $S$. The input to FAQ is a set of functions and the output is a function computed using a series of aggregates over the variables and input functions. More specifically, for each $i \in [n]$, let $X_i$ be a variable on some discrete domain $\text{Dom}(X_i)$, where $|\text{Dom}(X_i)| \geq 2$. The FAQ problem is to compute the following function

$$\varphi(x_{|F|}) = \Theta^{(f+1)}_{x_{f+1} \in \text{Dom}(X_{f+1})} \cdots \Theta^{(n)}_{x_n \in \text{Dom}(X_n)} \psi_S(x_S), \quad (1)$$

where

- $\mathcal{H} = (V, E)$ is a multi-hypergraph. $V = [n]$ is the index set of the variables $X_i$, $i \in [n]$. Overloading notation, $V$ is also referred to as the set of variables.

- The set $F = \{f\}$ is the set of free variables for some integer $0 \leq f \leq n$. Variables in $V \setminus F$ are called bound variables. (Free and bound are logic terminology. Free variables are group-by variables in database nomenclature.)

- $D$ is a fixed domain, such as $\{0, 1\}, \mathbb{R}^+, \mathbb{Z}$.

- For every hyperedge $S \in \mathcal{E}$, $\psi_S : \prod_{i \in S} \text{Dom}(X_i) \rightarrow D$ is an input function (also called a factor).

- For every bound variable $i > f$, $\Theta^{(i)}$ is a binary (aggregate) operator on the domain $D$.

- And, for each bound variable $i > f$ either $\Theta^{(i)} = \circ$ or $(D, \Theta^{(i)}, \circ)$ forms a commutative semiring (with the same 0 and 1). Informally, this means that we can do addition and multiplication over $D$ and still remain in the same set.

If $\Theta^{(i)} = \circ$, then $\Theta^{(i)}$ is called a product aggregate; otherwise, it is a semiring aggregate. (We assume that there is at least one semiring aggregate.)

Because for $i > f$ every variable $X_i$ has its own aggregate $\Theta^{(i)}$ over all values $x_i \in \text{Dom}(X_i)$. in the rest of the paper we will write $\Theta^{(i)}$ to mean $x_i \in \text{Dom}(X_i)$.

We will refer to $\varphi$ as an FAQ-query. We use FAQ-SS to denote the special case when there is a Single Semiring aggregate, i.e. $\Theta^{(i)} = \circ, \forall i > f$, and $(D, \circ, \circ)$ is a semiring [6].

**Example 5. (Aggregate query in RDBMS) Consider** the following query over relations $R(a, b), S(a, c), T(b, c, d, e), U(d, f), V(e, f), W(e, g), Y(f, h)$, where all attributes are integers.

**SELECT R.b, U.d, SUM(W.e)**

**FROM R, S, T, U, V, W, Y**


We now explain how the above query can be reduced to an FAQ instance. Relation $R(a, b)$ is modeled with a function $\psi_R(a, b) \rightarrow \{0, 1\}$, where $\psi_R(a, b) = \text{true}$ iff $(a, b) \in R$. Similarly, we can think of relations $S, T, U, V, Y$ as functions $\psi_S, \psi_T, \psi_U, \psi_V, \psi_Y$, with $\{0, 1\}$ values. We single out one relation $W(e, g)$ where the modeling is different: $\psi_W(e, g) = 1$ if $(e, g) \in W$ and 0 otherwise. The corresponding FAQ-query is

$$\varphi(d, b) = \sum_a \sum_c \sum_e \sum_f \sum_g \sum_h \psi_R(a, b) \psi_S(b, c) \cdots \psi_Y(b, h) \psi_W(b, d)$$

(For readability, we did not write the argument lists of the functions $\psi_R, \psi_S, \ldots$ etc. They should be obvious from context.) Note that a tuple in the output of the aggregate query has the schema $(b, d, \varphi(b, d))$. The corresponding hypergraph is shown in Fig. 1a. The set of free variables is $F = \{b, d\}$. The domain is $D = \mathbb{Z}$, the set of integers. Note

A triple $(D, \oplus, \circ)$ is a commutative semiring if $\oplus$ and $\circ$ are commutative binary operators over $D$ satisfying the following: (1) $(D, \oplus, \circ)$ is a commutative monoid with an additive identity, denoted by 0. (2) $(D, \oplus)$ is a commutative monoid with a multiplicative identity, denoted by 1. (3) $\circ$ distributes over $\oplus$. (4) For any element $e \in D$, $e \circ 0 = 0 \circ e = 0$. SIGMOD Record, March 2017 (Vol. 46, No. 1)}
also that the above reduction to FAQ still works if we replace sum by another aggregate, e.g., max.

In order to explain later the connection of InsideOut to query rewriting, we also write the above query in LogiQL, an extension of Datalog supported by the LogicBlox engine [7]:

\[ Q(b, d) = s \leftarrow \text{agg} < \text{cs} = \text{total}(e) > R(a, b), S(a, c), T(b, c, d, e), U(d, f), V(e, f), W(e, g), Y(f, h). \]

In the above, \( \text{agg} \) is short for aggregate, \( \text{total} \) is equivalent to \( \text{sum} \) in SQL, the notation \( Q(b, d)\Rightarrow s \) means that the head predicate is \( Q(b, d, s) \) where \( (b, d) \) is a key, hence the query computes \( Q(b, d, \text{sum}(e)) \).

Now that we have established the scope of FAQ, in the remainder of this paper we show a perhaps surprising result that an FAQ problem can be solved by one simple yet efficient algorithm. The algorithm can be implemented as a set of ordinary database queries. The runtime matches or improves upon the best known runtimes in many application areas that the FAQ framework captures. The runtime depends on the order of variable aggregates in the FAQ expression, which naturally leads us to the question of how to re-order those aggregates to obtain the best runtime without changing the semantic meaning of the expression.

2. THE INSIDEOUT ALGORITHM

Parts of this section will be familiar to readers who have been exposed to elementary graphical models [24]. There are, however, a couple of ideas that are taken from new developments in database theory [29, 28, 3] that are likely not known in the graphical model literature. For each factor \( \psi_S \), define its size to be the number of non-zero points under its domain: \( |\psi_S| := \{ x_S | \psi_S(x_S) \neq 0 \} \).

Basic variable elimination. To describe the intuition, we first explain InsideOut as it applies to the special case of FAQ-SS (or SumProd). The idea behind variable elimination [17, 37, 36] is to “fold” common factors, exploiting the distributive law:

\[
\bigoplus_{x_f} \cdots \bigoplus_{x_n} \psi_S(x_S) = \bigoplus_{x_f} \cdots \bigoplus_{x_{n-1}} \psi_S(x_S) \otimes \bigoplus_{x_n \in \partial(n)} \psi_S(x_S),
\]

where the equality follows from the fact that \( \otimes \) distributes over \( \oplus \). \( \partial(n) \) denotes all edges incident to \( n \) in \( H \) and \( U_n = \cup_{S \in \partial(n)} S \). Assume for now that we can somehow efficiently compute the intermediate factor \( \psi_{U_n - \{n\}} \). Then, the resulting problem is another instance of FAQ-SS on a modified multi-hypergraph \( H' \), constructed from \( H \) by removing vertex \( n \) along with all edges in \( \partial(n) \), and adding back a new hyperedge \( U_n - \{n\} \). Recursively, we continue this process until all variables \( X_n, \ldots, X_{f+1} \) are eliminated. Textbook treewidth-based results for PGM inference are obtained this way [24]. In the database context (i.e. given an FAQ-query over the Boolean semiring), the intermediate result \( \psi_{U_n - \{n\}} \) is essentially an intermediate relation of a query plan, the folding technique exploiting distributive law corresponds to “pushing the aggregate down” the query plan [13].

Introducing the indicator projections. While correct, basic variable elimination as described above is potentially not very efficient for sparse input factors, i.e. factors where the number of non-zero entries is much smaller than the product of the active domain sizes. This is because the product that was factored out of the scope of \( X_n \) might annihilate many entries of the intermediate result \( \psi_{U_n - \{n\}} \), while we have spent so much time computing \( \psi_{U_n - \{n\}} \). For example, for an \( S \notin \partial(n) \) such that \( S \subseteq U_n \) and tuple \( y_S \) such that \( \psi_S(y_S) = 0 \), we do not need to compute the entries \( \psi_{U_n - \{n\}}(x_{U\setminus\{n\}}) \) for which \( y_S = x_S \); those entries will be eliminated later anyhow. The idea is then to only compute those \( \psi_{U_n - \{n\}}(x_{U\setminus\{n\}}) \) values that will “survive” the other factors later on. One simple way to achieve this would be to compute, for each \( S \in E - \partial(n) \), an “indicator factor” that checks if \( \psi_S(x_S) \) is 0 or not. Formally, for any two sets \( T \subseteq S \), and a given factor \( \psi_S \), the function \( \psi_{S/T} : \prod_{i \in T} \text{Dom}(X_i) \rightarrow D \) defined by

\[
\psi_{S/T}(x_T) := \begin{cases} 1 & \exists x_{S-T} \text{ s.t. } \psi_S(x_T, x_{S-T}) \neq 0 \\ 0 & \text{otherwise} \end{cases}
\]

is called the indicator projection of \( \psi_S \) onto \( T \). Using indicator factors, InsideOut computes the following factor when marginalizing \( X_n \) away:

\[
\psi_{U_n - \{n\}}(x_{U\setminus\{n\}}) = \bigoplus_{x_n} \left( \bigotimes_{S \in \partial(n)} \psi_S \right) \left( \bigotimes_{S \notin \partial(n), S \cup \{n\} \neq \emptyset} \psi_{S/S \cap U_n} \right)
\]

(2)
Another minor tweak is the observation that, if there is a hyperedge $S \in \mathcal{E} - \partial(n)$ for which $S \subset U_n$, then we do not use the indicator projection $\psi_{S/\partial(n)}$: we can use $\psi_S$ itself to compute the intermediate factor $\psi_{U_n - \{n\}}$, and then remove $\psi_S$ from $\mathcal{H}'$.

Example 6. We explain how the ideas above are implemented in Example 5. First, the order in which we choose to eliminate variables might have a huge effect on the runtime. For now, let us assume that we somehow decided to rewrite $\varphi(b, d)$ using the following variable order, where we trace the first couple of steps of the InsideOut algorithm without the indicator projection: (Example 10 later explains how this order is related to the tree decomposition in Fig. 1b.)

$$\varphi(b, d) = \sum_{c} \sum_{a} \sum_{e} \sum_{f} \psi_R \psi_S \psi_T \psi_v \psi_w \psi_y \psi_z$$

$$= \sum_{c} \sum_{a} \sum_{e} \sum_{f} \psi_R \psi_S \psi_T \psi_v \psi_w \psi_y \psi_z$$

$$= \sum_{c} \sum_{a} \sum_{e} \sum_{f} \psi_R \psi_S \psi_T \psi_v \psi_w \psi_y \psi_z$$

$$= \sum_{c} \sum_{a} \sum_{e} \sum_{f} \psi_R \psi_S \psi_T \psi_v \psi_w \psi_y \psi_z$$

$$= \sum_{c} \sum_{a} \sum_{e} \sum_{f} \psi_R \psi_S \psi_T \psi_v \psi_w \psi_y \psi_z$$

The first two steps are straightforward, where we eliminated $g$ and $h$. In LogiQL, these intermediate factors are computed with the following two rules

$$\text{psi1}[f] = s1 <- \text{agg}[s1 = \text{count()}] >> Y(f,h)$$

$$\text{psi2}[e] = s2 <- \text{agg}[s2 = \text{total(e)}] >> W(e,g)$$

The mathematical abstraction corresponds to rewriting a query into a series of smaller queries. Next, we explain how the ideas above are implemented in LogiQL rules.

$$\varphi(b, d) = \sum_{c} \sum_{a} \sum_{e} \psi_R \psi_S \psi_T \psi_v \psi_w \psi_y \psi_z$$

$$= \sum_{c} \sum_{a} \sum_{e} \psi_R \psi_S \psi_T \psi_v \psi_w \psi_y \psi_z$$

leading to the following LogiQL rules

$$\text{proj3}(c) <- \text{T}(a,b,c,d,e)$$

$$\text{proj4}(d,c) <- \text{S}(a,c)$$

$$\text{psi1}(a,b,c) = s1 <- \text{agg}[s1 = \text{total}(a)] >> \text{psi3}(e,f) \text{ if } \text{agg}[s1 = \text{count()}]$$

$$\text{psi2}(e,f) = s2 <- \text{agg}[s2 = \text{total}(e)] >> \text{psi4}(c,h) \text{ if } \text{agg}[s2 = \text{count()}]$$

$$\text{psi3}(e,f) = \text{psi4}(c,h)$$

Note that $\psi_4$ has values in $\{0, 1\}$ although $\psi_4$ can have any value in $\mathbb{Z}$. The final LogiQL rules are

$$\text{proj4}(b,c) <- \text{psi4}(b,c,d)$$

$$\text{psi4}(b,c,d) = s4$$

The general FAQ problem. The above strategy does not care if the variable aggregates where the same or different: As long as $(D, Q^{(n)})$ is a semiring, we can fold the common factors and eliminate $X_n$. Thus, InsideOut works almost as is for a general FAQ instance (as opposed to FAQ-SS).

$$\varphi(x, y) = \sum_{z} \sum_{w} \psi_R(x, y, z, w)$$

We are left with an FAQ-instance whose hypergraph is exactly $H' = H - \{n\}$: the hypergraph obtained form $H$ by removing vertex $n$ from the vertex set and all incident hyperedges. The sub-problems are of the form of product marginalizations of individual factors $\psi_S$ for $S \in \partial(n)$, each of which can be computed in linear time in $|\psi_S|$. The product marginalization step is algorithmically much easier because it does not create the intermediate factor $\psi_{U_n - \{n\}}$. As for $S \notin \partial(n)$, we replace $\psi_S$ by the power factor $\psi_S^{\partial(n)} = \psi_S(\mathcal{X}_n)^{\partial(n)}$, which can be done in linear time with a log $|\mathcal{X}_n|$ blowup using the repeated squaring algorithm. Note the key fact that this power is with respect to the product aggregate $\odot$. In most (if not all) applications of FAQ, there is one additional property: most of the time, $\odot$ is an idempotent operator over the active domain. For example, in the #QCQ problem $\odot$ is the usual product operator and the domain that it aggregates over is $\{0, 1\}$ (before there is a sum outside). In this case, $\psi_S^{\partial(n)} = \psi_S$. 
\((\psi_S(x_S))_{\operatorname{Dom}(x_S)} = \psi_S(x_S)\), and we do not need to spend the linear nor log-blupup time. For more details on product idempotence, see \cite{2}.

**FAQ sub-problems as natural joins.** In the above we have explained how InsideOut breaks a big problem into smaller problems. In the product marginalization case, the sub-problems are easy to solve: they can be solved in linear time. The most difficult problems, however, are of the form (2). This is exactly an FAQ-query where we marginalize out only one variable, with the remaining variables free. Zooming in, problem (2) is of the form

\[
\psi_{u_n - \{n\}}(x_{u_n - \{n\}}) = \bigoplus_{x_n \in F \subset E_n} \psi_F,
\]

where \(H_n = (U_n, E_n)\) is the sub-FAQ-query hypergraph. The problem is solved by computing \(\psi_{u_n}(x_{u_n}) = \bigotimes_{F \in E_n} \psi_F\) first. Once the \(\psi_{u_n}\) is computed, marginalizing away \(x_n\) to obtain \(\psi_{u_n - \{n\}}\) is trivial.

Computing the inner product is a natural join problem in disguise. Each input factor \(U_F\) is represented using a table of tuples of the form \([x_S, \psi_S(x_S)]\). Essentially, \(x_S\) is the (compound) key and \(\psi_S(x_S)\) is the value in this relation. Again, recall that entries not in the table have \(\psi_S\)-value 0. Hence, to compute \(\psi_{u_n}\), we can first join the tables \(U_F\) using only the key space. For each tuple \(x_{u_n}\) in the result of this join, we record the value \(\psi_{u_n}(x) = \bigotimes_{F \in E_n} \psi_F(x_F)\). The runtime is dominated by the natural join’s runtime.

**Worst-case optimal join algorithms.** Computing the natural join is a very well-studied problem with exciting new developments in the past decade or so. There are new worst-case optimal algorithms \cite{35, 28, 29, 1} that operate quite differently from traditional query plans, in the sense that they no longer compute one pairwise join at a time, but instead process the query globally. While the vast majority of database engines today still rely on traditional query plans, new, complex data analytics engines are switching to worst-case optimal algorithms: LogicBlox’s engine \cite{7} is built on a worst-case optimal algorithm called LeapFrog Triejoin \cite{35} (LFTJ), and the Myria data analytics platform supports a variant of LFTJ \cite{12}.

We briefly outline these results here. The generic form of the natural join problem can be posed in our hypergraph language as \(Q = \bigcap_{F \in E} R_F\), where \(H = (V, E)\) is the query hypergraph. The vertices of this hypergraph consist of all attributes. Each hyperedge \(F \in E\) corresponds to an input relation \(R_F\) whose attributes are \(F\). The natural join problem can be thought of as a constraint satisfaction problem: each input relation \(R_F\) imposes a constraint where a tuple \(x_F\) satisfies the constraint if \(x_F \in R_F\). A tuple \(x\) on all variables \(V\) is an output of the join if the projection \(x_F\) satisfies \(R_F\) for all \(F \in E\).

LFTJ \cite{35} can be viewed as backtracking-search algorithm, which was known some 50 years ago in the AI and constraint programming world \cite{16, 20}. (In contrast, by saving intermediate results \(\psi_{u_n - \{n\}}\) instead of re-computing them each time, InsideOut can be thought of as *dynamic programming*. The duality between backtracking search and dynamic programming is well-known \cite{33}.) LFTJ fixes some variable ordering \(X_1, \ldots, X_n\) of the query \(Q\), then performs “leap-frogging” to find the first binding \(x_1\) that does not yet violate any constraints \(R_F\); once \(x_1\) is found, it looks for the first binding \(x_2\) such that the partial tuple \((x_1, x_2)\) does not violate any constraint. The algorithm proceeds this way until either a full binding \(x\) is constructed in which case \(x\) is an output, or no good binding is found. For example, if no feasible binding for \(x_3\) is found, then the algorithm backtracks to the next good binding of \(x_2\).

The first advantage of backtracking search is that it requires only \(O(1)\)-extra space: it does not cache any computation. The second advantage, amazingly, is that a join algorithm based on back-tracking search such as LFTJ or others in \cite{28,29} are worst-case optimal, in the sense that the algorithm runs in time bounded by the worst-case output size. To state the output size bound, we need the following notion. Define the *fractional edge cover polytope* \(P(H)\) associated with a hypergraph \(H\) to be the set of all vectors \(\lambda = (\lambda_F)_{F \in E}\) satisfying the following linear constraints:

\[
\lambda \geq 0, \quad \text{and} \quad \sum_{F \in E} \lambda_F \geq 1, \quad \forall F \in V.
\]

A vector \(\lambda \in P(H)\) is called a *fractional edge cover* of \(H\). The join output size is bounded above by \(\sum_{F \in E} |R_F|^{\lambda_F}\), for any \(\lambda \in P(H)\). The best bound \(AGM(H)\), known as the AGM-bound \cite{8, 21}, is obtained by solving the linear program

\[
\min \left\{ \sum_{F \in E} \lambda_F |R_F| : \lambda \in P(H) \right\}.
\]

**Example 7.** Consider the query computing \(\psi_S\) in Example 6. The join query on the keys has the following shape:

\[Q = U(d, f) \Join V(e, f) \Join I(f) \Join J(e) \Join K(d, e)\]

Then, \(AGM(Q) = |U|^{1/2} |V|^{1/2} |I|^{1/2} |J|^{1/2} |K|^{1/2}\), where \(\lambda\) is a fractional edge cover of the query’s hypergraph. Suppose all input relations have the same size \(N\), then the optimal bound is obtained by setting \(\lambda_d = \lambda_e = \lambda_f = 1/2\), and \(\lambda_d = \lambda_e = \lambda_f = 0\). Worst-case optimal algorithms run in time \(\tilde{O}(N^{3/2})\) for this instance. Any traditional join-tree based plan runs in \(\Omega(N^2)\)-time for some input \cite{28}. Moreover, without the indicator projection of \(I(b, c, d, e)\), there would be no \(K(d, e)\) above, the best edge cover would be \(\lambda_d = \lambda_e = \lambda_f = 1\), and the runtime would become \(\Omega(N^3)\).

**Runtime analysis.** Let \(N\) denote the input size, \(|\text{output}|\) the output size, and \(K\) the set of \(k \in |n|\) for which \(\ominus(k) \neq \ominus\) (note that \(\ominus(f) \subseteq K\)). Also, let \(AGM(Q_k)\) denote the AGM-bound on the \(k\)th sub-query’s hypergraph \(H_k\). Then, it is not hard to show \cite{2} that the runtime of InsideOut is

\[
\tilde{O}\left( N + \sum_{k \in K} AGM(H_k) + |\text{output}| \right).
\]

The first term is input-preprocessing time, second is the total subproblem solving time, and third is the unavoidable output reporting time. From (4), we can write down a precise expression for the runtime of InsideOut. Minimizing the resulting (somewhat complicated) expression leads to the dynamic programming algorithm for the MCM problem and the FFT algorithm for the DFT (see \cite{2} for details).

In the above discussion, we assumed that variables were eliminated in order \(X_n, X_{n-1}, \ldots, X_1\). However, there is no reason to force InsideOut to follow this particular order. In particular, there might be a different variable ordering for which expression (4) is a lot smaller and the algorithm still works correctly on that ordering (see \cite{2}). This is where the main technical contributions of our work in \cite{2} begin. We need to answer the following two fundamental questions:

**Question 1.** How do we know which variable orderings are equivalent to the original FAQ-query expression?
Question 2. How do we find the “best” variable ordering among all equivalent variable orderings?
In the next two sections, we sketch how we answered the above two questions and followup questions in theory and in practice.

3. THEORETICAL CONTRIBUTIONS

To answer the above questions, we start with some definitions. A variable ordering \( \sigma \) is \( \varphi \)-equivalent if permuting the variable aggregates of \( \varphi \) using \( \sigma \) gives an expression \( \varphi' \) that is semantically-equivalent to \( \varphi \), i.e., that always returns the same output as \( \varphi \) no matter what the input is. Let \( \text{EVO}(\varphi) \) denote the set of all \( \varphi \)-equivalent variable orderings.

**Example 8.** The FAQ query \( \varphi' \) below is \( \varphi \)-equivalent.

\[
\varphi = \sum_a \sum_b \max_c \psi_1(a, b) \psi_2(a, c) \psi_3(c, d),
\]
\[
\varphi' = \sum_a \sum_b \max_c \psi_1(a, b) \psi_2(a, c) \psi_3(c, d).
\]

This is because \( \varphi \) can be written as

\[
\varphi = \sum_a \left( \left( \sum_c \psi_2(a, c) \psi_3(c, d) \right) \max_b \psi_1(a, b) \right).
\]

Now, for any \( \sigma \in \text{EVO}(\varphi) \), let \( \mathcal{H}_\sigma^k \) denote the \( k \)th sub-query’s hypergraph when we run \text{InsideOut} on \( \sigma \). Ideally, we would like to find \( \sigma \) minimizing the expression \( \sum_k \text{AGM}(\mathcal{H}_\sigma^k) \). However, this expression is data-dependent and thus it is a bit difficult to handle in a mathematically clean way. We simplify our objective by approximating the bound (4) a little: we upperbound \( \text{AGM}(\mathcal{H}_\sigma^k) \) by the fractional edge cover number of the subgraph \( \mathcal{H}_\sigma^k \), i.e. \( \text{AGM}(\mathcal{H}_\sigma^k) \leq N^n(\mathcal{H}_\sigma^k) \), where \( \rho'(\mathcal{H}_\sigma^k) \) is upperbounded by \( \overline{\rho} \left( N^{\max_{k \in K} \rho'(\mathcal{H}_\sigma^k)} + \text{output} \right) \). InsideOut on variable ordering \( \sigma \) runs in \( \tilde{O}(N^{\max(\rho') + \text{output}}) \)-time, where \( \text{faqw}(\sigma) \): (1) hurdles one might face in a practical implementation of \text{InsideOut}, and (2) whether practical implications are as good as what the theory says.

**Example 9.** In Example 8, the original order in \( \varphi \) has an FAQ-width of 2, because eliminating \( c \) first corresponds to joining \( \psi_2 \) and \( \psi_3 \) in time \( \Omega(N^2) \). In contrast, the order in \( \varphi' \) has an FAQ of 1, allowing to evaluate \( \varphi \) in time \( O(N) \).

To solve the optimization problem (5), the first problem we have to address is to precisely characterize the set \( \text{EVO}(\varphi) \). Our approach, sketched in Fig. 2, is to construct an expression tree of the FAQ query \( \varphi \). The expression tree defines a partially ordered set on the variables called the precedence poset. Then, to completely the characterization of EVO, we show that every ordering in EVO is component-wise equivalent (CWE) to a linear extension of LinEx(\( P \)). (See [2] for details.) Thus, if we do not care about query complexity, we can take the orange path in Fig 2 and bruteforce compute an optimal variable ordering \( \sigma' \), run it through InsideOut, for a total runtime of \( O(N^{\max(\rho') + \text{output}}) \).

However, in some FAQ-framework’s applications such as in graphical models, we cannot simply sweep query complexity under the rug. Moreover, computing the \( \text{faqw} \) is NP-hard because \( \text{faqw} \) is a strict generalization of the fractional hypertree width (fhtw), which is NP-hard [19] to compute. Hence, we find a good approximation for the \( \text{faqw} \). This is the green path in Fig. 2: from the expression tree, we construct a tree decomposition for \( \mathcal{H} \); then, from the GYO-elimination order of this tree decomposition we obtain a variable ordering \( \sigma \) for which we can show that \( \text{faqw}(\sigma) \leq \text{faqw}(\sigma^*) + g(\text{faqw}(\sigma^*)) \), where \( g \) is any known approximation of \( \text{fhtw} \) (the best of which is due to Marx [20]).

**Example 10.** The variable ordering used earlier in Example 6 is a GYO-elimination order for the tree decomposition in Fig. 1b. (In GYO, when we eliminate variable \( X_n \), the set \( U_n \) becomes a bag of the tree decomposition whose children are the bags corresponding to \( \partial(n) \).) In particular, bags \( \{f, h\}, \{e, g\}, \) and \( \{d, e, f\} \) resulted from eliminating \( h, g, \) and \( f \) respectively. The tree decomposition has width \( \text{fhtw} = 3/2 \), same as the \( \text{faqw} \) of the variable ordering.

From these ideas, we obtained many corollaries, some of which are summarized in Table 1. The results in Table 1 span three areas: (1) CSPs and Logic; (2) PGMs and (3) Matrix operations. Except for joins, problems in area (1) need the full generality of FAQ formulation, where \text{InsideOut} either improves upon existing results or yields new results. Problems in area (2) can already be reduced to FAQ-SS. Here, \text{InsideOut} improves upon known results since it takes advantage of Grohe and Marx’s more recent fractional hypertree width bounds. Finally, problems in area (3) of Table 1 are classic. \text{InsideOut} does not yield anything new here, but it is intriguing to be able to explain the textbook dynamic programming algorithm for Matrix-Chain Multiplication [15] as an algorithm to find a good variable ordering for the corresponding FAQ-instance. The DFT result is a re-writing of Aji and McEliece’s observation [6].

4. PRACTICAL IMPLICATIONS

In this section we address two questions the readers might have regarding \text{InsideOut}: (1) hurdles one might face in a practical implementation of \text{InsideOut}, and (2) whether practical implications are as good as what the theory says.

**Additional hurdles and how to solve them.** There are a couple of problems we have to solve to implement \text{InsideOut} effectively.

The first problem is, in real-world queries, we do not just have materialized predicates as inputs, we also have predicates such as \( a < b, a + b = c, \) negations and so on. These predicates do not have a “size.” To solve this problem, one solution is to set the “size” of those predicates to be \( \infty \) when computing the AGM-bound. For instance, if we have a subquery of the form \( Q \leftarrow R(a, b), S(b, c), a + b = c \), then for setting the size of \( a + b = c \) to be infinite, \( \text{AGM}(Q) = N^2 \).

This solution does not work for two reasons. (1) If we knew \( a + b = c \), then it is easy to infer that \( |Q| \leq N \) and also to compute \( Q \) in time \( O(N) \): scan over tuples in \( R \), use \( a + b = c \) to compute \( c \), and see if \( (b, c) \in S \). In other words, the AGM-bound is no longer tight. (2) This solution may give an \( \infty \)-bound when the output size is clearly bounded. Consider, for example, the query \( Q \leftarrow R(a), S(b), a + b = c \); in this case, \( a, b, c \) is the only hyperedge covering vertex \( c \) in the fractional edge cover. Our implementation at LogicBlob...
Figure 2: Sketch of main technical contributions

Table 1: Runtimes of algorithms assuming optimal variable ordering is given. Problems shaded red are in CSPs and logic (D = {0, 1} for CSP and D = N for #CSP), problems shaded green fall under PGMs (D = R1), and problems shaded blue fall under matrix operations (D = C). N denotes the size of the largest factor (assuming they are represented with the listing format). htw(ϕ) is the notion of integral cover width for PGM. PW(H) is the optimal width of a prefix graph of H and DM(H) = poly(F-ss(H), fhtw(H)), where F-ss(H) is the [f]-quantified star size. Z is the output size in listing representation. Our width FAQ(ϕ) is never worse than any of the three and there are classes of queries where ours is unboundedly better than any of the three and there are classes of queries where ours is unboundedly better than any of the three. In DFT, N = p^m is the length of the input vector. ˜O hides a factor of poly(|H|) · log N.

makes use of generalizations of AGM to queries with functional dependencies and immaterialized predicates (such as a + b = c). These new bounds are based on a linear program whose variables are marginal entropies [4, 5].

The second problem is to select a good variable ordering to run InsideOut on. In principle, one does not have to use the AGM-bound or the bounds from [4, 5] to estimate the cost of an FAQ subquery. If one were to implement InsideOut inside any RDBMS, one could poll that RDBMS’s optimizer to figure out the cost of a given variable ordering. However, there are n! variable orderings, and optimizer’s cost estimation is time-consuming. Furthermore, some subqueries have inputs which are intermediate results. Hence, it is much faster to compute a variable ordering minimizing the FAQ of the query, defined on the bounds in [4, 5]. As the problem is NP-hard, either an approximation algorithm [3] or a greedy heuristic suffices in our experience.

InsideOut is bottom-up dynamic programming. We can also solve FAQ queries with top-down (memoized) dynamic programming. In hindsight, this was the approach that Bak—
ibavey et al. [9] and Otteleau and Závodný [32] took to solve FAQ over a single semiring. We can also limit the amount of memoization in a top-down strategy to attain performance gain in some cases [22].

**Practical Impact.** It is trivial to construct classes of queries on real datasets for which InsideOut-style of algorithms gives arbitrarily large speedups over traditional RDBMSs. In fact, even when dynamic programming does not take effect, the speedup of backtracking search (and thus worst-case optimal algorithms) over traditional query plans is already huge [30].

The impact of the FAQ-framework and the InsideOut algorithm, however, go much beyond these toy queries (even when run on real datasets).

InsideOut is a component of LogicBlox’s effort to extend LogiQL to be a probabilistic programming language [10], as part of DARPA’s PPAML and MUSE programs. The component the algorithm handles is inference in discrete graphical models.

Learning from the beautiful work of Otteleau and Schleich [31, 34], we realized [27] that InsideOut can be used to train a large class of machine learning models inside the database. Our implementation showed orders of magnitude speedup over the traditional data modeler route of exporting the data and running it through R or Python. These models are trained with different variations of gradient descents, whose (pre-)computation steps are FAQ queries. What is much more interesting than the vanilla FAQ framework we presented above is that, in these applications, we want to compute many (in the 100K-range) FAQ queries all at once, making dynamic programming much more crucial to the performance. Another related approach was considered in [25].

5. **CONCLUDING REMARKS**

The FAQ framework showed that many common computational tasks over a very wide range of domains such as CSPs, machine learning, relational database, logic, and matrix computations can be performed inside a database using the same abstraction. The main idea is to blur the line between data and computation, as we use the database to store, compute, and process functions. The glue of the framework is a simple dynamic programming algorithm called InsideOut, which can be cast as a query-rewriting method and thus it is readily implementable within any RDBMS. These ideas are implemented, tested, and validated within the LogicBlox database system. Our theory predicts practice very well, which is a beautiful thing to see.

6. **REFERENCES**


